Finding Classes in Legacy Code Using Cluster Analysis

Arie van Deursen
Tobias Kuipers

CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
{arie, kuipers}@cwi.nl, http://www.cwi.nl/~{arie,kuipers}/

1 Introduction

Old software systems are still in use because they implement useful business tasks. Unfortunately, they are difficult to adapt. They have been subject to maintenance repeatedly, have become less and less comprehensible, and they are closely tied to old technology.

To address these problems, much research has been carried out to extract business objects — groups of data and associated operations — from existing systems. These business objects are extracted by inspecting sources, documentation, or by asking the original developers to provide appropriate design information; see, e.g., [6, 2] for techniques supporting this process. Once they are extracted, they form the basis for an object-oriented re-implementation of the kernel of the system, a re-implemntation which thanks to the object-orientation is far more flexible and easier to adapt.

There are three important steps in the extraction of business objects: (1) identification of potential instance variables; (2) identification of potential methods; (3) grouping of variables and methods into classes. In this paper, we concentrate on the third step, for which we propose the use of *cluster analysis*, a general technique for finding groups in data [4].

In this paper we will show that cluster analysis can support the extraction of an object oriented re-design in that it can

- group instance variables into classes based on the common use of global in the legacy source code;
- assign methods to classes based on the use of variables in the procedures as occurring in the legacy code;
- visualize the common use of variables and procedures in the legacy system.

	P1	P2	P3	P4
NAME	1	0	0	0
TITLE	1	0	0	0
INITIAL	1	0	0	0
PREFIX	1	0	0	0
NUMBER	0	0	0	1
NUMBER-EXT	0	0	0	1
ZIPCD	0	0	0	1
STREET	0	0	1	1
CITY	0	1	0	1

Figure 1: The usage matrix that is used as input for the cluster analysis

In the next section, we describe our current experience with using cluster analysis for the extraction of business objects. We are currently experimenting with a 100 KLOC COBOL system and, although the research is still in a preliminary stage, have been able to find some useful groupings. In the final section we list issues that require further discussion or future research.

Acknowledgements The authors were sponsored by bank ABN Amro, software house DPFinance, and the Dutch Ministery of Economical Affairs via the Senter Project #ITU95017 “SOS Resolver”. Our results are based on work done jointly with the authors of [9, 10].

2 Tools and Experiments

When trying to find classes in legacy code, our current emphasis is on *data*, i.e., variables. Because we can perform cluster analysis on a set of variables and their distribution throughout the legacy code, we first need to establish such a set. Obviously, we need to discriminate between important and unimportant variables. We do this by looking at *data persistency* (whether data is written to file), and by human inspection.

For each of the variables in the set, we determine their location throughout the code. We determine whether or not a specific variable is used in a particular part of the code. In our experiments, we determine the usage of variables both per program, and per section (of a COBOL program). The result of this operation is a matrix, such as the one in Table 1. Here we have 9 variables, distributed over 4 programs. Each entry in the matrix shows whether a variable is used in a program (1) or not (0).

Because we want to perform cluster analysis on this data, we need to calculate some form of distance (or Hamming metric) between the variables. If we see the rows of the matrix as vectors, then each variable occupies a position in a four dimensional space. We can now calculate a distance between any two variables, using vector arithmetic.

For instance, the distance between NAME and CITY from the table in Figure 1 is

$$|\overline{\mathrm{NAME}} - \overline{\mathrm{CITY}}| = \left| \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} - \begin{pmatrix} 0 \\ 1 \\ 0 \\ 1 \end{pmatrix} \right| = \left| \begin{pmatrix} 1 \\ -1 \\ 0 \\ -1 \end{pmatrix} \right| =$$

$$\sqrt{1^2 + (-1)^2 + 0 + (-1)^2} = \sqrt{3}$$

This distance is called the *Euclidean* distance. Many other distance measures are possible.

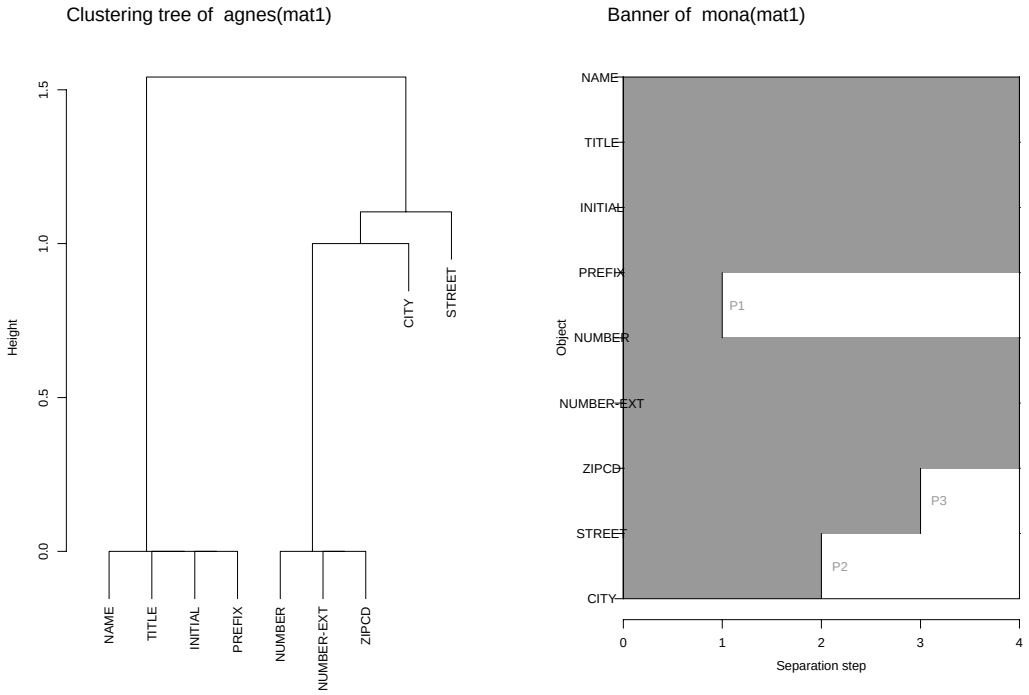
If we put the distances between any two variables in a matrix, we get a so-called distance, or *dissimilarity* matrix. (The greater the distance, the bigger the dissimilarity). Such a dissimilarity matrix can be used as input to a clustering algorithm. We used an hierarchical clustering algorithm (AGNES, from [4]). This algorithm starts by putting each element in its own cluster, and then proceeds by creating new clusters that contain two (or more) clusters that are closest to one another. Finally, only one cluster remains, and the algorithm terminates. All intermediate clusterings can be seen as branches on a tree, in a dendrogram. Figure 2(a) shows the dendrogram that results from clustering the data in Figure 1.

Because our matrix only contains boolean values (either a variable is used in a program, or it is not), we can also apply a different cluster algorithm. The algorithm MONA [4] does not take a dissimilarity matrix as input, but works directly on the matrix from Figure 1. It tries to split all the variables into groups by looking whether certain attributes are true or false. Figure 2(b) shows the result of this analysis. It shows that the variables NAME, TITLE, INITIAL, and PREFIX are separated from the others because of attribute P1. CITY was then separated from the resulting variables because of attribute P2, and so on. In this case, Figures 2(a) and 2(b) show the same clustering.

Although this example uses only a fraction of the code that we are experimenting with, we can clearly see two main clusters: one containing information about a persons name, and the other about a persons address. The address is further divided in a cluster containing STREET, one containing CITY, and one containing NUMBER, NUMBER-EXT, and ZIPCD. This is particularly nice, because the combination of a zip-code, a number and a number extension, uniquely defines an address in The Netherlands. Note that ordering between CITY and STREET is completely arbitrary. They both have the same distance to both other clusters.

3 Discussion

We have used the clustering techniques as described in the previous section to split the central data structure of a COBOL legacy system into smaller substructures. This central structure is a single record containing 45 fields, described in a copy book included in most programs of the system. The clustering performed on this record resulted in



(a) Dendrogram representation

(b) Banner representation

Figure 2: The resulting clustering from Figure 1

a proposal of 10 classes which was perceived as meaningful by the original system developers.

These initial results are encouraging, and show that clustering indeed can play a role in the grouping of variables and visualization of variable use. There are, however, still a number of important issues that require further elaboration and discussion.

First of all, we still have to investigate how our method complements or can be combined with other methods. Of particular interest is the relationship with *concept analysis* [8], the use of clustering to measure module cohesion [3, 1], and the use of clustering for subsystem classification [5, 7].

Secondly, the clustering can be used to assign programs to groups of variables as well, as illustrated by the program names occuring in Figure 2(b). More work is needed to understand the role of additional information, such as call dependencies, to steer the clustering process.

Furthermore, the data set as clustered on in the experiments conducted so far, might be changed in a number of ways:

- Every textual occurrence of a variable whose postfix matches one of the data item names is counted as a “use”. More refined versions are possible, taking data flow into account, and giving different weights to for example variable updates,

operator arguments, assignments without operators, etc.

- The notion of "proximity" can be refined: at this moment, two variables are related if they occur in the same program. Alternative groupings involve occurrence on the same control flow path, use in the same procedure, etc.
- The experiments suggest that it makes little difference whether to take frequencies of variable occurrences into account. Can this phenomenon be explained?
- The experiments use a very large data set, which includes a fairly significant amount of noise. It may be possible to improve the clusters found by drastically reducing the data set.

References

[1] B. Durnota and C. Mingins. Tree-based coherence metrics in object-oriented design. In C. Mingins, B. Heabich, J. Potter, and B. Meyer, editors, *Technology for Object-Oriented Languages and Systems; TOOLS 9*, pages 489–504. Prentice-Hall, 1993.

[2] H. Fergen, P. Reichelt, and K. P. Schmidt. Bringing objects into COBOL: MOORE - a tool for migration from COBOL85 to object-oriented COBOL. In *Proceedings of the Conference on Technology of Object-Oriented Languages and Systems (TOOLS 14)*, pages 435–448. Prentice-Hall, 1994.

[3] D. H. Hutchens and V. R. Basili. System structure analysis: Clustering with data analysis. *IEEE Transactions on Software Engineering*, SE-11(8):749–757, 1985.

[4] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley, 1990.

[5] A. Lakhotia and J. M. Gravley. Toward experimental evaluation of sybsystem classification recovery techniques. In *Second Working Conference on Reverse Engineering; WCRE95*, pages 262–269. IEEE Computer Society, 1995.

[6] C. L. Ong and W. T. Tsai. Class and object extraction from imperative code. *Journal of Object-Oriented Programming*, pages 58–68, March–April 1993.

[7] W. Pedrycz and J. Waletzky. Fuzzy clustering in software reusability. *Software—Practice and Experience*, 27(3):245–270, 1997.

[8] M. Siff and T. Reps. Identifying modules via concept analysis. In *nternational Conference on Software Maintenance, ICSM97*. IEEE Computer Society, 1997. To appear.

[9] T. Wiggerts. Using clustering algorithms in legacy systems remodularization. In *4th Working Conference on Reverse Engineering*. IEEE Computer Society, 1997. To appear.

[10] T. Wiggerts, H. Bosma, and E. Fielt. Scenarios for the identification of objects in legacy systems. In *4th Working Conference on Reverse Engineering*. IEEE Computer Society, 1997. To appear.

Each clustering algorithm comes in two variants: a class, that implements the fit method to learn the clusters on train data, and a function, that, given train data, returns an array of integer labels corresponding to the different clusters. For the class, the labels over the training data can be found in the labels_ attribute. Input data. One important thing to note is that the algorithms implemented in this module can take different kinds of matrix as input. All the methods accept standard data matrices of shape [n_samples, n_features]. These can be obtained from the classes in the sklearn.f @article{Fuhr2011UsingDA, title={Using Dynamic Analysis and Clustering for Implementing Services by Reusing Legacy Code}, author={Andreas Fuhr and Tassilo Horn and Volker Riediger}, journal={2011 18th Working Conference on Reverse Engineering}, year={2011}, pages={275-279} }. Andreas Fuhr, Tassilo Horn, Volker Riediger. Published in 18th Working Conference on Reverse Engineering 2011.Â Migrating legacy systems towards Service-Oriented Architectures requires the identification of legacy code that is able to implement the new services. This paper proposes an approach combining dynamic analysis and data mining techniques to map legacy code to business processes and to identify code for service implementations based on this mapping. Exam- features in legacy code, and the results of preliminary ex- ples of such idioms include multiple inheritanceâ€“ through perimentation. language built-in constructs or through design idiomsâ€“ del- egation, or, using an ad-hoc implementation where no spe- Aspect-oriented development, aspect mining, refactor- ciï¬c care is taken to package the functional feature. Our ing, concept analysis, functional aspects. strategy for identifying functional features relies, at least in part, on the detection of such programming and design id- 1 Introduction ioms in legacy code.Â To ï¬nd out how several functional features interweave within the same class hierarchy, consider the way a devel- e.g. [1]).Â Aspect mining using event traces. In Finally, formal concept analysis (FCA)